

DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9747

NEW COMPUTATIONAL RESULTS ON THE DISCRETE TIME/COST TRADE-OFF PROBLEM IN PROJECT NETWORKS

by

E. DEMEULEMEESTER

B. DE REYCK

B. FOUBERT

W. HERROELEN

M. VANHOUCKE



Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

ONDERZOEKSRAPPORT NR 9747

**NEW COMPUTATIONAL RESULTS ON THE DISCRETE
TIME/COST TRADE-OFF PROBLEM IN PROJECT
NETWORKS**

by

E. DEMEULEMEESTER

B. DE REYCK

B. FOUBERT

W. HERROELEN

M. VANHOUCKE

NEW COMPUTATIONAL RESULTS ON THE DISCRETE TIME/COST TRADE-OFF PROBLEM IN PROJECT NETWORKS

**Erik DEMEULEMEESTER • Bert DE REYCK • Bram FOUBERT⁺ •
Willy HERROELEN • Mario VANHOUCKE**

December 1997

Operations Management Group
Department of Applied Economics
Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven (Belgium)
Phones: 32-16-32 69 72, 32-16-32 69 66, 32-16-32 69 70, Fax 32-16-32 67 32
E-mail: <first name>.<name>@econ.kuleuven.ac.be

⁺ Currently at UFSIA, Antwerp.

This research was supported by the National Science Foundation under Research Contract FWO N° G.0131.96

NEW COMPUTATIONAL RESULTS ON THE DISCRETE TIME/COST TRADE-OFF PROBLEM IN PROJECT NETWORKS

ABSTRACT

We describe a new exact procedure for the discrete time/cost trade-off problem in deterministic activity-on-the-arc networks of the CPM type, where the duration of each activity is a discrete, nonincreasing function of the amount of a single resource committed to it. The objective is to construct the complete and efficient time/cost profile over the set of feasible project durations. The procedure uses a horizon-varying approach based on the iterative optimal solution of the problem of minimizing the sum of the resource use over all activities subject to the activity precedence constraints and a project deadline. This optimal solution is derived using a branch-and-bound procedure which computes lower bounds by making convex piecewise linear underestimations of the discrete time/cost trade-off curves of the activities to be used as input for an adapted version of the Fulkerson labelling algorithm for the linear time/cost trade-off problem. Branching involves the selection of an activity in order to partition its set of execution modes into two subsets which are used to derive improved convex piecewise linear underestimations. The procedure has been programmed in Visual C++ under Windows NT and has been validated using a factorial experiment on a large set of problem instances.

Keywords: Project management; CPM; Time/cost trade-off; Branch-and-bound.

INTRODUCTION

The specific problem addressed in this paper is the discrete time/cost trade-off problem in project networks of the CPM type. The specification of a project is assumed to be given in activity-on-the-arc (AoA) notation by a directed acyclic graph (dag) $D = (N, A)$ in which N is the set of nodes, representing network "events", and A is the set of arcs, representing network "activities". We assume, without loss of generality, that there is a single start node 1 and a single terminal node n , $n = |N|$. The duration y_a of activity $a \in A$ is a discrete, nonincreasing function $g_a(x_a)$ of the amount of a single resource allocated to it; i.e., $y_a = g_a(x_a)$. The pair y_a, x_a shall be referred to as a "mode", and shall be written as: (y_a, x_a) . Thus an activity that assumes four different durations according to four possible resource allocations to it shall be said to possess four modes.

Three possible objective functions have been studied in the literature (De et al.¹). For the first objective function (referred to as problem 1, $Tcpm, disc, mulC_{max}$ in the classification scheme of Herroelen et al.²) a limit R is specified on the total availability of a single nonrenewable resource type. The problem is then to decide on the vector of activity durations (y_1, \dots, y_m) , $m = |A|$, that completes the project as early as possible under the limited availability of the single nonrenewable resource type. A second objective function (referred to as problem 1, $Tcpm, \delta_n, disc, mulav$) reverses this problem formulation: now there is a limit δ_n on the project length and we try to minimize the sum of the resource use over all activities. For the third objective function (referred to as problem 1, $Tcpm, disc, mulcurve$) the complete time/cost trade-off function for the total project is to be computed, i.e., all the efficient points (T, R) such that with a resource limit R a project length T can be obtained and such that no other point (T', R') exists for which both T' and R' are smaller than or equal to T and R .

The early contributions to the basic time-cost trade-off problem in CPM networks (Moder et al.³) assumed ample resource availability and hence did not explicitly take resource decisions into account. A direct activity cost function was used instead, representing the direct activity costs as a function of activity duration. Activity durations are bounded from below by the crash duration (corresponding to a maximum allocation of resources) and bounded from above by the normal duration (corresponding to the most efficient resource allocation). Essentially, the project costs correspond to a requirement for a nonrenewable resource, the total requirement of which is to be minimized. This corresponds to minimizing the (required) availability of the resource. When the activity cost functions are linear, the problem is denoted as $Tcpm, \delta_n, lin, mulav$ in the classification scheme of Herroelen et al.². The problem can be solved optimally by the well-known Fulkerson labelling algorithm⁴.

While the problem has been widely studied under the assumption of continuous time/cost relationships (Moder et al.³), the literature on the discrete case has been rather sparse. De et al.⁵ have shown that the discrete time/cost trade-off problem is NP-hard under the three objectives mentioned above. De et al.¹ offer an excellent review of the literature. Demeulemeester et al.⁶ report on promising computational experience with two exact procedures. The first algorithm is based on a procedure for finding the minimal number of reductions necessary to transform a general network to a series-parallel

network. The second algorithm minimizes the computational effort in enumerating alternative modes through a branch-and-bound search tree.

The objective of this paper is to present and validate a new optimal procedure for problem $1, Tlcpm, disc, mulcurve$. The paper is organized as follows. In Section 2 we clarify the solution methodology. Section 3 provides an illustrative problem example. Computational experience is reported in Section 4. A last section then offers overall conclusions.

AN EXACT PROCEDURE

The solution procedure presented in this paper provides an optimal solution to problem $1, Tlcpm, disc, mulcurve$ using a horizon-varying approach which involves the iterative optimal solution of problem $1, Tlcpm, \delta_n, disc, mulav$ (minimize the sum of the resource use over all activities subject to a project deadline) over the feasible project durations in the interval bounded from below by \underline{t}_n (the project duration obtained under activity crash modes) and from above by \bar{t}_n (the project duration obtained under normal activity modes).

The branch-and-bound algorithm

Problem $1, Tlcpm, \delta_n, disc, mulav$ is solved using a branch-and-bound algorithm which is based on the following logic. The algorithm starts by computing for each activity a convex piecewise linear underestimation of its discrete time/cost trade-off curve as shown in bold lines in Figure 1. An initial lower bound is obtained by solving the resulting time/cost trade-off problem using a variant of the well-known Fulkerson labelling algorithm⁴ described below. This algorithm yields for each activity a mode (x°, y°) .

For each activity a *vertical distance*, vd_{ij} , is computed which measures the quality of the convex underestimation. Two rules may be used to perform this computation. *Rule 1* computes vd_{ij} as the minimum of the distances $vd1$ and $vd2$. $vd1$ is the distance between y° and the cost of the nearest mode to the left of x° on the convex piecewise linear underestimation (cost y_2 in Figure 1). $vd2$ is the distance between y° and the cost of the nearest mode to the right of x° on the convex piecewise linear underestimation (cost y_4 in Figure 1). *Rule 2* computes the vertical distance between y° and the curve covering all modes for the activity based on linear interpolation (distance $vd3$ in Figure 1).

Branching is done by identifying the activity with the largest vertical distance (*Rule 2* uses *Rule 1* as a tie-breaker) and partitioning its set of modes into two subsets. The first subset, set_1 , consists of the set of modes with a smaller duration than x° (modes (x_1, y_1) and (x_2, y_2) in Figure 1). The second subset, set_2 , consists of the set of modes with a duration greater than or equal to x° (modes (x_3, y_3) , (x_4, y_4) and (x_5, y_5) in Figure 1). These subsets are used to obtain two new convex piecewise linear underestimations for the activity. The first descendant node in the search tree replaces the current underestimation for the activity by the one provided by set_1 (see Figure 2(a)), the second descendant node replaces the current underestimation by the one provided by set_2 (see Figure 2(b)). The solution of

the two corresponding problems using the adapted Fulkerson algorithm yields the corresponding new lower bounds. Branching continues from the node with the smallest lower bound (arbitrary tie-break).

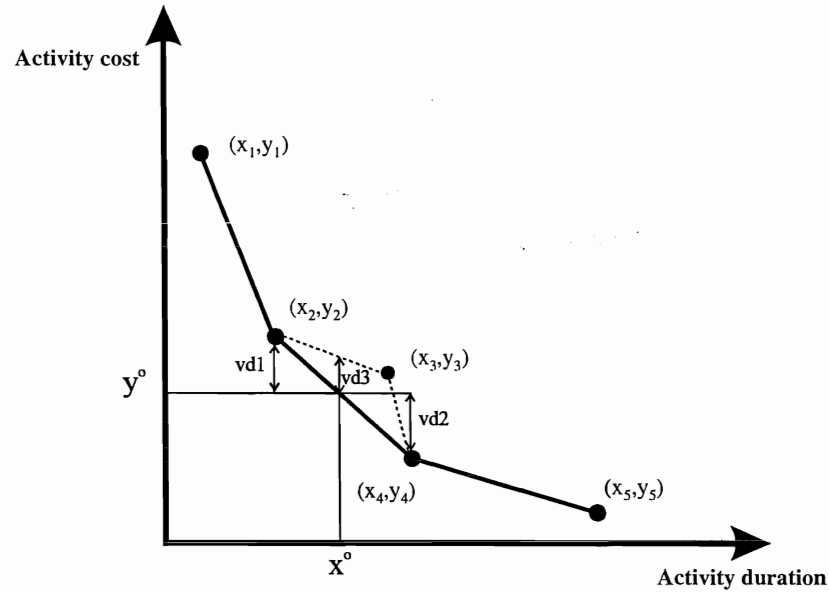


Figure 1. Convex piecewise linear underestimation of an activity's discrete time/cost trade-off profile

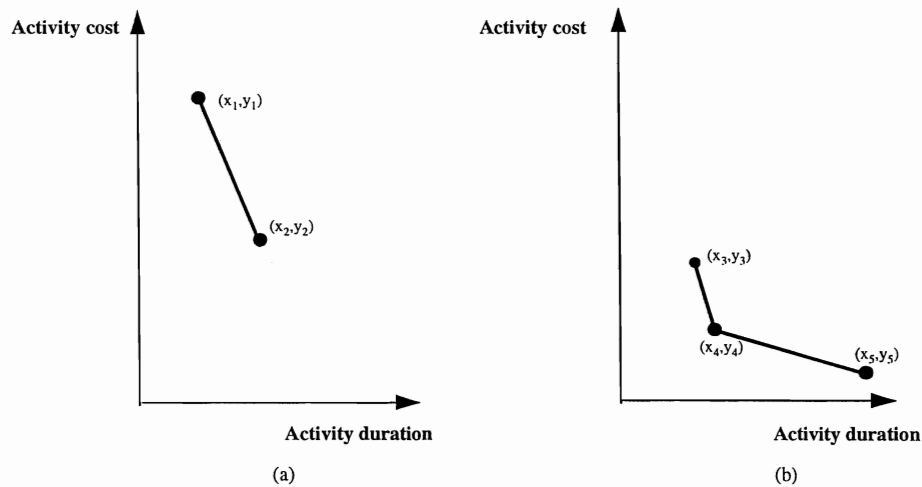


Figure 2. Two convex piecewise linear underestimations for the bottleneck activity

Backtracking occurs when the computed lower bound exceeds (or equals) the cost of an earlier found schedule or when problem $1, Tlcpm, \delta_n, disc, mulav$ becomes infeasible at a certain level of the search tree. The procedure stops when backtracking leads to the source node of the search tree.

Computing the lower bound

The original Fulkerson labelling algorithm⁴ for solving problem $1, Tlcpm, \delta_n, lin, mulav$ associated the flow capacity of an arc with the negative slope of the corresponding activity direct cost curve. The algorithm is adapted for convex piecewise linear trade-off curves as follows. Each time the duration of an activity (arc) is reduced to a value which coincides with a breakpoint in the piecewise

linear underestimating cost curve the arc's capacity has to be updated. The way this is done is best illustrated using the time/cost trade-off curve presented in Figure 3.

The breakpoints of the curve correspond to the modes (20,5), (16,7), (13,10) and (8,17). Each time the duration of the activity switches between the intervals $]\infty,8]$, $]8,13]$, $]13,16]$ and $]16,20]$ the corresponding arc capacity is updated. At the start of the algorithm, the duration of the activity will be initialized to its normal value (20), while the arc capacity is set equal to 0.5, the negative slope of the line segment connecting points (20,5) and (16,7). Suppose the activity's duration has been reduced to 16. It is clear that the marginal cost value of 0.5 is no longer valid as a further reduction in duration can only be obtained at a marginal cost of 1 (the negative slope of the line segment connecting points (16,7) and (13,10)). Therefore the (residual) capacity is increased by 0.5 (the difference between 1 and 0.5). Suppose the activity's duration is reduced to 8. Further reduction is impossible: the marginal cost is ∞ , the negative slope of the line segment connecting points (8,17) and (8, ∞). The arc capacity is set to ∞ . In a similar way, an increase in an activity duration would lead to a decrease in the corresponding (residual) arc capacity. Assume the duration of the activity is extended from 13 (where the curve shows a breakpoint) to 15. The (residual) capacity is now reduced by 0.4, the difference between 1.4 (the negative slope of the line segment connecting points (13,10) and (8,17)) and 1 (the negative slope of the line segment connecting points (16,7) and (13,10)).

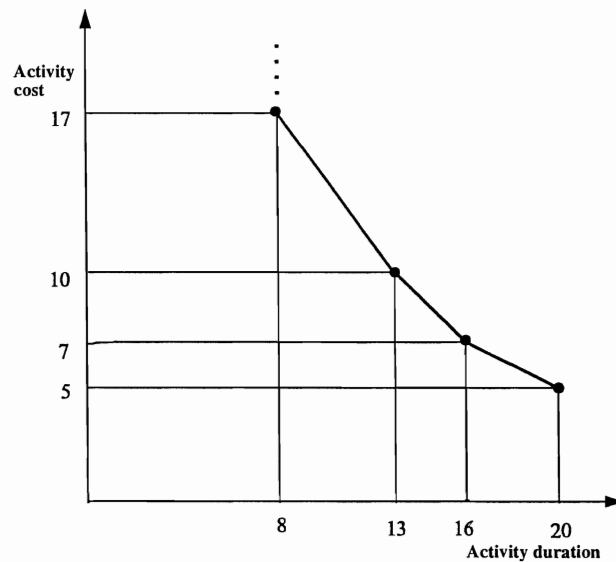


Figure 3. An underestimating convex piecewise linear time/cost trade-off curve.

The horizon-varying algorithm

The horizon-varying algorithm can now be described. The algorithm computes a convex piecewise linear underestimation for the time/cost trade-off profile using a set of durations MS_{ij} . The negative slopes of the line segments of the convex underestimation are saved in CC_{ij} , the convex set of marginal costs of activity (i,j) . If activity (i,j) only has one mode, $CC_{ij} = \{\}$. The durations which

correspond to the modes lying on the convex underestimating curve are saved in the set CD_{ij} , the convex set of durations of activity (i,j) .

Step 0: Deadline computation.

- Compute \bar{t}_n , the critical path length with every activity (i,j) at its normal duration.
- Compute \underline{t}_n , the critical path length with every activity (i,j) at its crash duration.
- Set the current project deadline $T = \bar{t}_n$.

Step 1: Initialisation.

- Let $ub(T) = \infty$, the upper bound of the project cost for deadline T .
- Set p , the level of the search tree, equal to 0.
- Compute for every activity (i,j) the convex piecewise linear underestimation with the set of durations $MS_{ij} = M_{ij}$, being the original set of possible durations d_{ij} for activity (i,j) .
- Run the adapted Fulkerson labelling algorithm to compute a lower bound lb and the corresponding activity durations w_{ij} .

Step 2: Identify the activity with the maximal vertical distance.

- Compute for each activity (i,j) its vertical distance vd_{ij} .
- Compute the maximal vertical distance $vd_{max} = \max_{(i,j) \in A} \{vd_{ij}\}$
- If $vd_{max} = 0$, the schedule is complete and feasible. Update $ub(T) = lb$ and go to step 4.
- Update the level of the branch-and-bound tree: $p = p + 1$.
- Store the activity (u,v) with $vd_{uv} = vd_{max}$ at level p (ties are broken arbitrarily). Store the corresponding sets MS_{uv} , CC_{uv} and CD_{uv} .

Step 3: Separate and branch.

- Generate two descendant nodes in the search tree. For the first node, define $set_1 = \{d_{uv} \in MS_{uv} \mid d_{uv} \geq w_{uv}\}$ and compute the convex piecewise linear underestimation with $MS_{uv} = set_1$. Store durations w_{ij} and lower bound lb_1 . For the second mode, define $set_2 = \{d_{uv} \in MS_{uv} \mid d_{uv} < w_{uv}\}$ and compute the convex piecewise linear underestimation with $MS_{uv} = set_2$. Store durations w_{ij} and lower bound lb_2 .
- Select the node with the smallest lower bound $lb = \min(lb_1, lb_2)$ for branching. If $lb \geq ub(T)$, go to step 4.
- Store the information for the remaining node r . Go to step 2.

Step 4: Backtracking.

- If the branching level $p = 0$, then go to step 5.
- If both nodes at level p have been evaluated, set $p = p - 1$ and repeat step 4.
- Evaluate the remaining node r at this level: if $lb_r \geq ub(T)$, set $p = p - 1$ and repeat step 4.
- Go to step 2

Step 5: Update project deadline.

- Set $T = T - 1$. If $T < \underline{t}_n$, stop; else, go to step 1.

AN ILLUSTRATIVE EXAMPLE

Consider the AoA network shown in Figure 4. The cost/duration profiles and the first convex piecewise linear underestimation for activities (1,2), (1,3), (2,4) and (3,4) are shown in Figures 5(a)-5(d). The original sets of durations are $M_{12} = \{11, 5\}$, $M_{13} = \{10, 6, 3, 1\}$, $M_{23} = \{0\}$, $M_{24} = \{5\}$ and $M_{34} = \{10, 8, 6, 4\}$. We illustrate the branch-and-bound procedure for a deadline $T = 14$. This will yield one point on the project's time/cost trade-off profile. The resulting branch-and-bound search tree is shown in Figure 6.

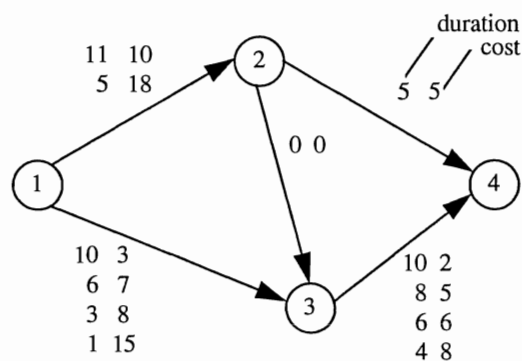


Figure 4. Project network example.

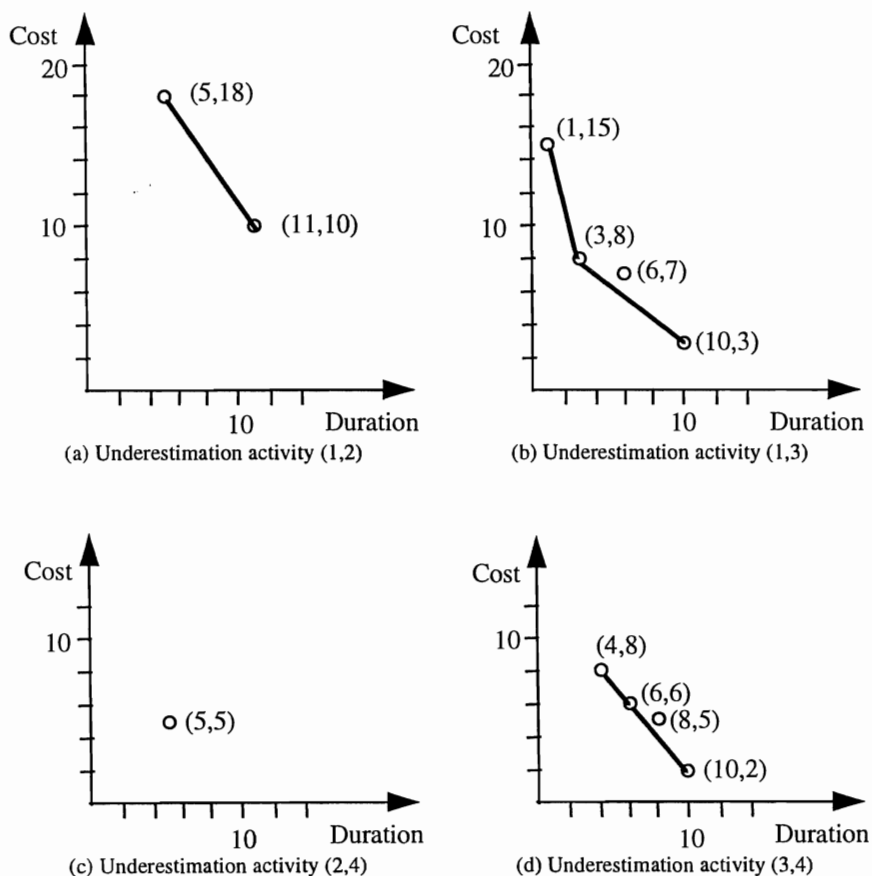


Figure 5. Underestimations for the activities of the example network

Step 1. Initialize $ub(T) = \infty$ and $p = 0$. Compute the convex piecewise linear underestimation with $MS_{ij} = M_{ij}$. This yields the convex set of durations $CD_{12} = \{11,5\}$, $CD_{13} = \{10,3,1\}$, $CD_{23} = \{0\}$, $CD_{24} = \{5\}$ and $CD_{34} = \{10,6,4\}$. The convex set of marginal costs is computed as $CC_{12} = \{1.33\}$, $CC_{13} = \{0.71,3.5\}$, $CC_{23} = \{\}$, $CC_{24} = \{\}$, and $CC_{34} = \{1,1\}$. The adapted Fulkerson algorithm yields a lower bound $lb = 28.4$ and corresponding activity durations $w_{12} = 9$, $w_{13} = 9$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 5$.

Step 2. Let us assume that the vertical distances are computed according to *Rule 1*: $vd_{12} = 2.67$, $vd_{13} = 0.71$, $vd_{23} = 0$, $vd_{24} = 0$, $vd_{34} = 1$. As an illustration, vd_{13} is computed as follows. The duration $w_{13} = 9$ corresponds to a cost on the piecewise linear underestimating curve of $3 + 0.71 * (10 - 9) = 3.71$. Then $vd_{13} = \min\{8 - 3.71; 3.71 - 3\} = 0.71$. The maximal distance is $vd_{12} = 2.67$. Activity (1,2) has the largest vertical distance. Store the corresponding information: $(u,v) = (1,2)$, $MS_{12} = \{11,5\}$, $CC_{12} = \{1.33\}$, $CD_{12} = \{11,5\}$. Update the branching level: $p = 1$.

It is interesting to observe that *Rule 2* would have computed the vertical distances as follows: $vd_{12} = 0$, $vd_{13} = 0.29$ (i.e., $4 - 3.71$), $vd_{23} = 0$, $vd_{24} = 0$, $vd_{34} = 0$. As a result, activity (1,3) would be identified as the activity with the maximal vertical distance.

Step 3. Generate the two descendant nodes at level $p = 1$ of the search tree. $set_1 = \{11\}$ and $set_2 = \{5\}$. Compute the underestimation with $MS_{12} = set_1 = \{11\}$, $CC_{12} = \{\}$ and $CD_{12} = \{11\}$. The adapted Fulkerson procedure yields $lb_1 = \infty$. Compute the underestimation with $MS_{12} = set_2 = \{5\}$, $CC_{12} = \{\}$ and $CD_{12} = \{5\}$. The adapted Fulkerson procedure yields $lb_2 = 32.6$ and $w_{12} = 5$, $w_{13} = 5$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 9$. Select the second node for branching.

Step 2. Compute the vertical distances: $vd_{12} = 0$, $vd_{13} = 1.43$, $vd_{23} = 0$, $vd_{24} = 0$, $vd_{34} = 1$. The maximal vertical distance is 1.43 for activity (1,3). Increase the branching level: $p = 2$. Store the corresponding information: $(u,v) = (1,3)$, $MS_{13} = \{10,6,3,1\}$, $CC_{13} = \{0.71,3.5\}$ and $CD_{13} = \{10,3,1\}$.

Step 3. Generate the two descendant nodes at level 2 of the search tree: $set_1 = \{10,6\}$ and $set_2 = \{3,1\}$. Compute the underestimation with $MS_{13} = \{10,6\}$, $CC_{13} = \{1\}$ and $CD_{13} = \{10,6\}$. The adapted Fulkerson procedure yields $lb_1 = 34$ and $w_{12} = 5$, $w_{13} = 6$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 8$. Compute the underestimation for $MS_{13} = \{3,1\}$, $CC_{13} = \{3.5\}$ and $CD_{13} = \{3,1\}$ and run the Fulkerson procedure: $lb_2 = 34$ and $w_{12} = 5$, $w_{13} = 3$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 9$. Select the first node for branching with $lb = 34$ (arbitrary tie-break).

Step 2. Compute the vertical distances: $vd_{12} = 0$, $vd_{13} = 0$, $vd_{23} = 0$, $vd_{24} = 0$, $vd_{34} = 2$. Activity (3,4) is the current activity with the maximal vertical distance. Store the corresponding information: $(u,v) = (3,4)$, $MS_{34} = \{10,8,6,4\}$, $CC_{34} = \{1,1\}$ and $CD_{34} = \{10,6,4\}$. Update the level of the search tree: $p = 3$.

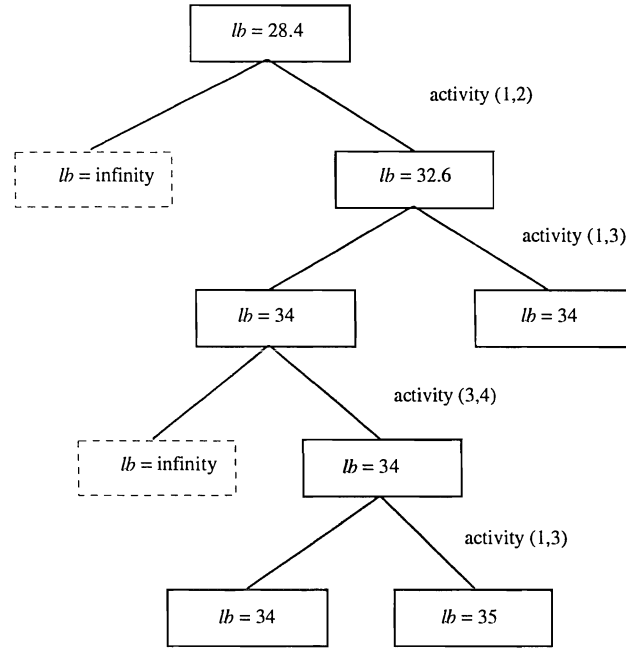


Figure 6. Branch-and-bound search tree for the problem example.

Step 3. Generate the two descendant nodes at level $p = 3$: $set_1 = \{10\}$ and $set_2 = \{8,6,4\}$. Compute the underestimation for $MS_{34} = \{10\}$, $CC_{34} = \{\}$ and $CD_{34} = \{10\}$ and use the Fulkerson algorithm:

$lb_1 = \infty$. Compute the underestimation for $MS_{34} = \{8,6,4\}$, $CC_{34} = \{0.5,1\}$ and $CD_{34} = \{8,6,4\}$ and use the Fulkerson routine to find $lb_2 = 34$ and $w_{12} = 5$, $w_{13} = 8$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 6$. Select the second node for branching with $lb = 34$.

Step 2. Compute the vertical distances: $vd_{12} = 0$, $vd_{13} = 2$, $vd_{23} = 0$, $vd_{24} = 0$, $vd_{34} = 0$. The maximal vertical distance is 2 for activity (1,3). Increase the branching level: $p = 4$. Store the corresponding information: $MS_{13} = \{10,6\}$, $CC_{13} = \{1\}$ and $CD_{13} = \{10,6\}$.

Step 3. Generate the two descendant nodes at level $p = 4$: $set_1 = \{10\}$ and $set_2 = \{6\}$. Underestimate for $MS_{13} = \{10\}$: $CC_{13} = \{\}$ and $CD_{13} = \{10\}$ and use the Fulkerson procedure to find: $lb_1 = 34$ and $w_{12} = 5$, $w_{13} = 10$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 4$. Underestimate for $MS_{13} = \{6\}$, $CC_{13} = \{\}$ and $CD_{13} = \{6\}$ and run the Fulkerson routine to find: $lb_2 = 35$ and $w_{12} = 5$, $w_{13} = 6$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 8$. Select the first node for branching.

Step 2. The maximal vertical distance is 0: update $ub = 34$.

Step 4. Restore the second node at level $p = 4$: $lb = 35 \geq ub$. Backtrack to level $p = 3$.

Step 4. Restore the first node at level $p = 3$: $lb = \infty \geq ub$. Backtrack to level $p = 2$.

Step 4. Restore the second node at level $p = 2$: $lb = 34 \geq ub$: backtrack to level $p = 1$.

Step 4. Restore the first node at level $p = 1$: $lb = \infty \geq ub$: backtrack to level $p = 0$.

Step 4. $p = 0$; repeat this procedure with $T = 13$.

COMPUTATIONAL RESULTS

The horizon-varying algorithm has been coded in Visual C++ Version 4.0 under Windows NT 4.0. In order to validate the algorithm we used *ProGen* (Kolisch et al.⁷) to generate 1,800 test instances in activity-on-the-node format using the parameter settings in Table I. Using five settings for the number of activities, six settings for the number of execution modes, two settings for the problem scale (activity duration and activity cost values), and three settings for the coefficient of network complexity (*CNC*, defined as the number of precedence relations divided by the number of activities (Pascoe⁸)), we obtained 180 problem classes, each consisting of 10 instances. These instances were transformed into activity-on-the-arc instances with minimum complexity index value (*CI*, defined as the number of node reductions sufficient - along with series and parallel reductions - to reduce a two-terminal directed acyclic network to a single edge) using the procedure described in De Reyck and Herroelen⁹.

Table I. Parameter settings used to generate the test instances

Number of activities	10, 20, 30, 40 or 50
Number of modes	fixed at 2, 4 or 6 or randomly chosen from the interval [1,3], [1,7], or [1,11]
Activity durations and costs	randomly selected from the interval [1,20] or [1,100]
Coefficient of network complexity	1.5, 1.8 or 2.1

Each problem instance was solved using the horizon-varying approach equipped with the first rule (referred to as *P1*) and the second rule (referred to as *P2*) for computing the vertical distance. The results obtained using the exact procedure based on finding the minimum number of node reductions necessary to transform the network to a series-parallel network (referred to as *P3*), developed by Demeulemeester et al.⁶, are used as a benchmark. *P3* is one of the best procedures currently available. Moreover, preliminary tests on several of the instances revealed that it outperformed the alternative procedure developed by Demeulemeester et al.⁶, which is based on minimizing the computational effort in enumerating alternative modes through a branch-and-bound tree. Each procedure was allowed to run on a Dell personal computer (Pentium 133 MHz processor) for a maximum CPU time of 200 seconds on each instance.

Table II represents the average CPU time (seconds), the average number of nodes generated in the search tree (only where relevant, i.e. for *P1* and *P2*), and the average percentage of problems solved optimally by each of the three procedures. The row labelled "All instances" gives the average results over all 1800 problem instances. Both horizon-varying procedures clearly outperform procedure *P3* for both performance measures. The best results are obtained by procedure *P1*. The four remaining rows

give more detailed results. The results in the row labelled “Number of activities” show that the three procedures solve all the 10-activity problems very quickly. Procedures *P1* and *P2* find the optimal solution for almost all 20-activity and most of the 30-activity problems. Clearly, the higher the number of activities, the higher the average CPU time required and the smaller the percentage of problems solved.

The results in the row labelled “Number of modes” reveal the negative effect of the number of modes on the efficiency and effectiveness of the three procedures: the higher the number of modes, the more CPU time required and the smaller the percentage of problems solved optimally. *P1* and *P2* optimally solve all 2-mode problems in less than 5 seconds on the average. Moreover, the problems with a fixed number of modes are more difficult to solve by any of the three procedures than the instances where the number of modes is randomly selected from the three corresponding intervals.

Table II. Average CPU time, average number of nodes in the search tree and average % problems solved optimally

	Average CPU time (seconds)			Average number of nodes in the search tree		Average % of problems solved optimally		
	<i>P1</i>	<i>P2</i>	<i>P3</i>	<i>P1</i>	<i>P2</i>	<i>P1</i>	<i>P2</i>	<i>P3</i>
All instances	62.09	65.80	104.13	30,500	31,008	78 %	76 %	51 %
Number of activities								
10	0.34	0.39	0.02	1,359	1,434	100 %	100 %	100 %
20	19.17	22.95	70.01	18,472	20,249	99 %	98 %	73 %
30	58.00	65.13	122.99	35,813	37,404	86 %	83 %	42 %
40	105.40	110.44	151.73	48,281	48,290	61 %	57 %	28 %
50	127.56	130.10	175.91	48,575	47,660	41 %	40 %	14 %
Number of modes								
2	3.77	4.72	49.21	2,679	2,679	100 %	100 %	79 %
4	85.82	90.72	130.79	40,390	40,116	69 %	68 %	39 %
6	122.20	123.44	151.91	62,987	61,165	49 %	48 %	25 %
1 to 3	2.63	2.86	27.99	1,719	1,737	100 %	100 %	89 %
1 to 7	63.30	71.60	118.46	29,906	32,631	81 %	77 %	46 %
1 to 11	94.85	101.46	146.44	45,319	47,717	66 %	61 %	29 %
CNC								
1.5	53.55	57.32	77.73	33,732	34,107	82 %	80 %	65 %
1.8	61.33	65.23	114.18	28,402	29,218	78 %	76 %	46 %
2.1	71.40	74.86	120.49	29,366	29,698	73 %	71 %	43 %
Scale								
1 to 20	49.35	52.64	101.95	19,594	19,218	84 %	82 %	52 %
1 to 100	74.84	78.96	106.31	41,406	42,660	71 %	69 %	51 %

The results in the row labelled “CNC” indicate the adverse effect of the *CNC* on problem complexity: the higher the *CNC*, the more difficult the problem. The row labelled “Scale” shows the impact of the problem scale (activity duration and activity cost values). The larger the scale, the more difficult it is to solve the problem instance.

Tables III, IV and V show the interaction effects between the number of activities and the number of modes for each of the three procedures. The first number in each cell denotes the average required CPU time in seconds for solving the problems in the corresponding class, while the second number denotes the average percentage of problems solved optimally within the CPU time limit.

Table III. Interaction between the number of activities and the number of modes for *P1*

Number of modes Number of activities	2	4	6	1 to 3	1 to 4	1 to 11
10	0.01 100 %	0.24 100 %	1.13 100 %	0.01 100 %	0.20 100 %	0.46 100 %
20	0.33 100 %	10.73 100 %	60.67 95 %	0.11 100 %	7.60 100 %	35.60 100 %
30	1.61 100 %	71.39 93 %	156.44 40 %	0.29 100 %	47.55 98 %	70.74 85 %
40	5.32 100 %	153.46 48 %	192.75 10 %	2.00 100 %	104.89 73 %	173.96 37 %
50	11.56 100 %	193.27 5 %	200.00 0 %	10.76 100 %	156.24 35 %	193.50 7 %

Table IV. Interaction between the number of activities and the number of modes for *P2*

Number of modes Number of activities	2	4	6	1 to 3	1 to 4	1 to 11
10	0.01 100 %	0.37 100 %	1.24 100 %	0.01 100 %	0.23 100 %	0.48 100 %
20	0.35 100 %	14.48 100 %	67.87 92 %	0.12 100 %	9.07 100 %	45.81 95 %
30	1.85 100 %	86.05 87 %	155.54 38 %	0.29 100 %	65.39 92 %	81.62 83 %
40	6.55 100 %	158.52 47 %	192.56 10 %	2.06 100 %	120.60 62 %	182.35 25 %
50	14.83 100 %	194.19 5 %	200.00 0 %	11.81 100 %	162.71 33 %	197.06 3 %

P1 solves all two-mode problems to optimality in very promising CPU times. Slightly more than 5 seconds are needed on the average to solve the 40-activity problems and some 12 seconds are

needed on the average for the 50-activity problems (Table III). The effectiveness drops when the number of activities reaches 40 and the number of modes is 4 or more. Most of the 50-activity problems with 4 or more execution modes are, given the CPU time limit, beyond the capabilities of procedure *P1*. *P2* is a close runner-up (Table IV). *P3*, being very fast on the small 10-activity instances, runs into trouble on the 50-activity, 2-mode problems (slightly more than one third of the problems can be solved optimally, as shown in Table V). Using *P3*, problems with 4 modes become very difficult if not impossible to solve within the specified CPU limit, when the number of activities reaches beyond 30.

Table V. Interaction between the number of activities and the number of modes for *P3*

Number of modes Number of activities	2	4	6	1 to 3	1 to 4	1 to 11
10	0.00 100 %	0.01 100 %	0.06 100 %	0.00 100 %	0.01 100 %	0.06 100 %
20	0.04 100 %	75.21 82 %	159.47 27 %	0.02 100 %	44.74 92 %	140.55 37 %
30	6.46 100 %	178.76 15 %	200.00 0 %	0.63 100 %	160.49 27 %	191.57 8 %
40	101.62 62 %	200.00 0 %	200.00 0 %	21.72 98 %	187.02 10 %	200.00 0 %
50	137.93 33 %	200.00 0 %	200.00 0 %	117.56 48 %	200.00 0 %	200.00 0 %

It was mentioned already that the coefficient of network complexity, *CNC*, seems to have an effect on the complexity of a problem instance: the higher the *CNC*, the more complex the problem. De Reyck and Herroelen⁹ have shown, however, that it might be dangerous to attribute a lot of complexity explaining power to *CNC* in many activity network problem settings. Moreover, previous studies of the discrete time/cost trade-off problem (De Reyck and Herroelen⁹) have revealed a strong effect of the *CI* on the CPU time needed to solve a problem instance: the higher its *CI*, the harder it is to solve the problem.

Tables VI, VII and VIII show the interaction effect between the complexity index *CI* and the number of activities in terms of the average CPU time required to solve the problems in each corresponding class. The results reported here must be interpreted with sufficient care. As mentioned earlier, the problem generator *ProGen* does not allow for the generation of networks satisfying pre-set values of *CI*. We generated the networks in activity-on-the-node format and transformed them into activity-on-the-arc instances with minimal *CI*-value. This explains the rather fragmentary results. For a given number of activities, the three procedures show an increasing CPU time as the value of the *CI* goes up. This pattern seems to be more pronounced for procedure *P3*. This result, already confirmed by De Reyck and Herroelen⁹, is not a surprise as the *CI*-concept lies at the very heart of the solution logic of procedure *P3*.

Table VI. Interaction between *CI* and the number of activities for *P1*

<i>CI</i>	Number of activities	10	20	30	40	50
1		0.21				
2		0.24				
3		0.36				
4		0.35				
5		0.36	9.50			
6			11.19			
7			11.26			
8			11.44	46.04		
9			17.34	44.86		
10			24.59	43.04	127.19	
11			28.73		82.56	
12			19.29		90.42	
13				50.51	104.51	106.61
14				59.21		119.49
15				61.02		126.81
16				77.97		121.94
17				71.02	104.06	
18				74.47		
19					104.49	
20				100.81	109.49	135.46
21						126.30
22					118.40	125.35
23					114.50	135.55
24					124.25	126.21
25						136.02
26						126.58
27						138.88
28						135.24
29						134.47

Table VII. Interaction between *CI* and the number of activities for *P2*

<i>CI</i>	Number of activities	10	20	30	40	50
1		0.25				
2		0.28				
3		0.40				
4		0.40				
5		0.42	11.49			
6			13.04			
7			12.49			
8			16.73	51.71		
9			22.80	51.66		
10			29.21	51.67	131.76	
11			34.34		92.14	
12			20.95		96.54	
13				56.27	102.12	112.94
14				70.20		123.83
15				69.13		130.56
16				83.28		122.90
17				76.29	111.78	
18				81.03		
19					108.19	
20				97.32	115.38	136.25
21						128.49
22					116.21	127.32
23					119.66	136.01
24					130.13	128.37
25						139.97
26						128.71
27						140.73
28						136.31
29						135.78

Table VIII. Interaction between *CI* and the number of activities for *P3*

<i>CI</i>	Number of activities	10	20	30	40	50
1		0.00				
2		0.00				
3		0.01				
4		0.02				
5		0.11	5.51			
6			15.21			
7			41.42			
8			39.57	70.56		
9			83.06	102.79		
10			96.96	121.65	121.22	
11			95.81		123.28	
12			110.94		128.26	
13				133.73	133.60	136.68
14				133.80		136.89
15				134.06		137.22
16				134.04		141.12
17				136.40	141.61	
18				136.72		
19					160.84	
20				144.61	156.18	171.08
21						171.35
22					183.84	198.19
23					170.27	186.29
24					174.60	200.00
25						200.00
26						200.00
27						200.00
28						198.59
29						200.00

CONCLUSIONS

This paper reports on a new exact procedure for problem $1, Tlcpm, disc, mulcurve$, i.e. the discrete time/cost trade-off problem in deterministic activity-on-the-arc networks, where the activities are subject to finish-start precedence relations and where the duration of each activity is a discrete, nonincreasing function of the amount of a single nonrenewable resource committed to it. The objective is to construct the complete and efficient time/cost profile over the set of feasible project durations. The procedure uses a horizon-varying approach based on the iterative optimal solution of problem $1, Tlcpm, \delta_n, disc, mulav$, i.e. the problem of minimizing the sum of the resource use over all activities subject to the activity precedence constraints and a project deadline. This optimal solution is derived using a branch-and-bound procedure which computes lower bounds by making convex piecewise linear underestimations of the discrete time/cost trade-off curves of the activities to be used as input for an adapted version of the Fulkerson labelling algorithm for the linear time/cost trade-off problem. Branching involves the selection of an activity in order to partition its set of execution modes into two subsets which are used to derive improved convex piecewise linear underestimations. The procedure has been programmed in Visual C++ under Windows NT for use on personal computers and has been validated using a factorial experiment on a large set of problem instances.

The results obtained are encouraging. Both horizon-varying procedures clearly outperform procedure *P3* developed earlier by Demeulemeester et al.⁶ in terms of the average required CPU time and the average percentage of problems solved optimally. The best results are obtained by procedure *P1*. The three procedures optimally solve all the 10-activity problems at very small average CPU times. Procedures *P1* and *P2* find the optimal solution for almost all 20-activity and most of the 30-activity problems in acceptable time. Clearly, the number of activities and the number of modes have a negative effect on both the effectiveness and the efficiency of the procedures.

Although, the rather fragmentary results on the impact of the complexity index must be interpreted with sufficient care, it was found that for a given *CI*-value, all three procedures need more CPU time as the number of activities goes up. For a given number of activities, the three procedures also show an increasing overall pattern of CPU times as the value of the *CI* goes up. This pattern seems to be more pronounced for procedure *P3*. This result, already confirmed by Demeulemeester et al.⁶, is not a surprise as the *CI*-concept lies at the very heart of the solution logic of procedure *P3*.

References

- 1 De P, Dunne EJ, Ghosh JB, Wells CE (1995). The discrete time-cost tradeoff problem revisited. *European J Opl Res* **81**: 225-238.
- 2 Herroelen W, Demeulemeester E, De Reyck B (1997). A classification scheme for project scheduling problems. In: Weglarz J (ed.). *Handbook on Recent Developments in Project Scheduling*. Kluwer Academic Publishers, to appear.
- 3 Moder JJ, Phillips CR, Davis EW (1983). *Project Management with CPM, PERT and Precedence Diagramming* 3rd ed.. Van Nostrand Reinhold: New York.
- 4 Fulkerson DR (1961). A network flow computation for project cost curves. *Mgmt Sci* **7**: 167-179.
- 5 De P, Dunne EJ, Ghosh JB, Wells CE (1997). Complexity of the discrete time-cost tradeoff problem for project networks. *Ops Res* **45**: 302-306.
- 6 Demeulemeester EL, Herroelen WS, Elmaghraby SE (1996). Optimal procedures for the discrete time/cost trade-off problem in project networks. *European J Opl Res* **88**: 50-68.
- 7 Kolisch R, Sprecher A, Drexel A (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Mgmt Sci* **41**: 1693-1703.
- 8 Pascoe TL (1966). Allocation of resources - CPM. *Rev fr Rech opér* **38**: 31-38.
- 9 De Reyck B, Herroelen W (1996). On the use of the complexity index as a measure of complexity in activity networks. *European J Opl Res* **91**: 347-366.

